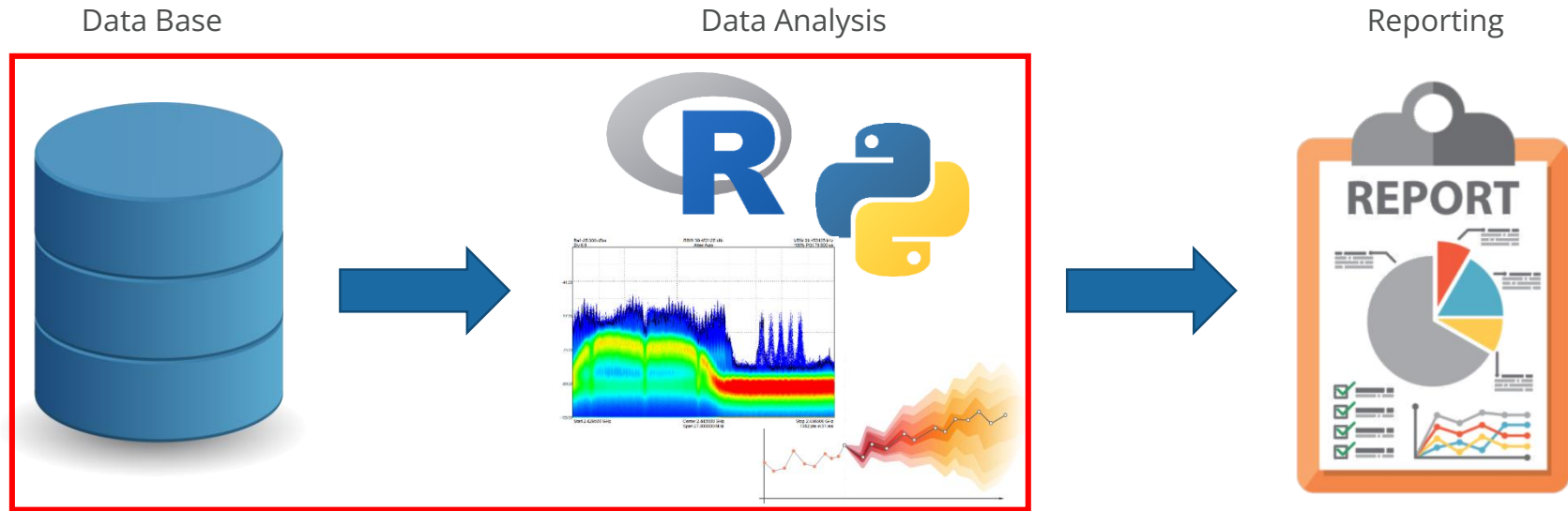


DB Analytics Support

Scalable Data Engineering

Motivation

Analytics Workflow



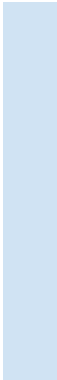
Outline

User Defined Structures

- Data Types
- Functions (UDF)
- Stored Procedures
- Aggregate Functions

Translating a Regression Model

- Regression
- Translating OLS



User Defined Structures

Define specific structures

- Encapsulate user specific functionality into a database object
- Bring data and application logic closer together
- Reduced network traffic
- Reusability of complex functionality
- Optimization of SQL statements
- Transactional protection

Implementation

- Data types
- Sequences
- Functions
- Procedures
- Aggregate Functions

Data Types

- `CREATE TYPE name AS`
 `( [ attribute_name data_type`
 `[ COLLATE collation ]] )`
- **attribute_name** - names of attributes to encapsulate in the new data type
- **collation** - specific sorting rules

```
CREATE TYPE customer_summary AS (  
    cid integer,  
    order_cnt integer  
);
```

```
CREATE TABLE R (a customer_summary);
```

a
(1, 17)
(2, 5)
(3, 29)
(4, 86)

```
SELECT (a).order_cnt FROM R WHERE (a).cid=2
```

order_cnt
5

Sequences

- `CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name`
 `[ AS data_type ]`
 `[ INCREMENT [ BY ] increment ]`
 `[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]`
 `[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]`
 `[ OWNED BY { table_name.column_name | NONE } ]`
- **TEMPORARY/TEMP** – only used during the current session, dropped on session exit
- **increment** – step with between calculated sequence values
- **start** – user defined start value
- **NO CYCLE** – restart sequence when all values are used

Sequences

- Example

```
CREATE SEQUENCE cust_id
AS integer
INCREMENT BY 2
START WITH 101;
```

```
CREATE TABLE comp_cust(
    cid integer DEFAULT nextval('cust_id'),
    c_name varchar
);
```

```
INSERT INTO comp_cust (c_name) VALUES
('Alice'),
('Bob'),
('Carl');
```

cid	c_name
101	Alice
103	Bob
105	Carl

Complex Data Types Definitions

- CREATE TYPE name (
 - INPUT = input_function,
 - OUTPUT = output_function
 - [, RECEIVE = receive_function]
 - [, SEND = send_function]
 - [, TYPMOD_IN = type_modifier_input_function]
 - [, TYPMOD_OUT = type_modifier_output_function]
 - [, ANALYZE = analyze_function]
 - [, SUBSCRIPT = subscript_function]
 - [, INTERNALLENGTH = { internallength | VARIABLE }]
 - [, PASSEDBYVALUE]
 - [, ALIGNMENT = alignment]
 - [, STORAGE = storage]
 - [, LIKE = like_type]
 - [, CATEGORY = category]
 - [, PREFERRED = preferred]
 - [, DEFAULT = default]
 - [, ELEMENT = element]
 - [, DELIMITER = delimiter]
 - [, COLLATABLE = collatable])

Complex Data Types Definitions

- Input and Output functions to parse and format in-/output strings
- Function called by the PostgreSQL cost model during plan optimization
- Storage strategy
- Delimiter character to be used between values in arrays made of this type

User Defined Structures

Function

- Mainly for calculations
- Return value
- No transactions or data manipulation
- Can be used in SQL queries
- Can be called by procedures

Procedure

- Execute a set of commands
- No return value
- Full transaction support and data manipulation (commit & rollback)
- Dedicated call mechanism
- Cannot be called by functions

User Defined Functions (UDF)

- `CREATE [ OR REPLACE ] FUNCTION`
 `name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [,`
 `...] ] )`
 `[ RETURNS rettype | RETURNS TABLE ( column_name column_type [, ...] ) ]`
 `{ LANGUAGE lang_name`
 `| PARALLEL { UNSAFE | RESTRICTED | SAFE }`
 `| COST execution_cost`
 `| AS 'definition'`
 `| AS 'obj_file', 'link_symbol'`
 `| sql_body`
 `...}`
- Choose language
- Parallel execution
- Input argument list

User Defined Functions (UDF)

- Example

```
CREATE OR REPLACE FUNCTION get_cust_sum(cid integer)
RETURNS customer_summary AS
$$
    SELECT o_custkey, count(o_orderkey)
    FROM orders
    WHERE o_custkey=cid
    GROUP BY o_custkey;
$$
LANGUAGE SQL;
```

```
select * from get_cust_sum(78002);
```

cid	Order_cnt
78002	9

```
select get_cust_sum(78002);
```

Get_cust_sum
(78002, 9)

User Defined Procedure

- `CREATE [ OR REPLACE ] PROCEDURE`
 `name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )`
 { `LANGUAGE lang_name`
 | `TRANSFORM { FOR TYPE type_name } [, ... ]`
 | `[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER`
 | `SET configuration_parameter { TO value | = value | FROM CURRENT`
 }
 | `AS 'definition'`
 | `AS 'obj_file', 'link_symbol'`
 | `sql_body`
 }
- Choose language
- Parallel execution
- Input argument list

User Defined Procedure

- Example

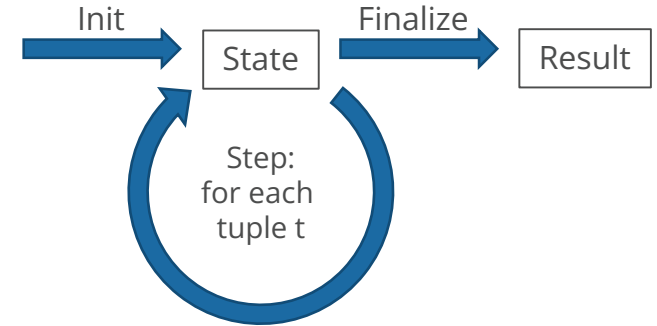
```
CREATE PROCEDURE insert_customer(name varchar)
LANGUAGE SQL
AS $$
    INSERT INTO comp_cust (c_name) VALUES (name);
$$;
```

```
CALL insert_customer('Dan');
CALL insert_customer('Earl');
```

cid	c_name
101	Alice
103	Bob
105	Carl
107	Dan
109	Earl

Aggregate Functions

- `CREATE AGGREGATE name (input_data_type [, ...])(
 SFUNC = sfunc,
 STYPE = state_data_type
 [, FINALFUNC = ffunc]
 [, INITCOND = initial_condition]
)`
- **INITCOND** – the initial condition of the aggregation
- **SFUNC** – step function calls for every tuple
- **FINALFUNC** – final function after all tuples are processed



Average:

State	sum:numeric	count:int
Init	$\text{sum} \leftarrow 0$	$\text{count} \leftarrow 0$
Step(t)	$\text{sum} \leftarrow \text{sum} + t.x$	$\text{count} \leftarrow \text{count} + 1$
Finalize	$\text{Result} \leftarrow \text{sum} / \text{count}$	

Aggregate Functions

```
CREATE FUNCTION taxi_accum (internal numeric, km numeric, ppk numeric)
RETURNS numeric AS
$$
    SELECT internal + km*ppk;
$$ LANGUAGE 'sql';
```

```
CREATE FUNCTION taxi_final(internal numeric)
RETURNS numeric AS
$$
    SELECT round(internal + 5, -1);
$$ LANGUAGE 'sql';
```

```
CREATE AGGREGATE taxi(km numeric, ppk numeric)
(
    INITCOND = 3.50,
    STYPE = numeric,
    SFUNC = taxi_accum,
    FINALFUNC = taxi_final
);
```

Taxi rides with several intermediate stops
3,50 for starting the ride
2,20 for each km
Rounding at the end to tip the driver

trip_id	km
1	3,4
1	5,3
1	2,9
2	9,3
2	1,6
2	4,3

```
SELECT trip_id,
       taxi(km, 2.20)
FROM   t_taxi
GROUP BY trip_id;
```



trip_id	km
2	40
1	30

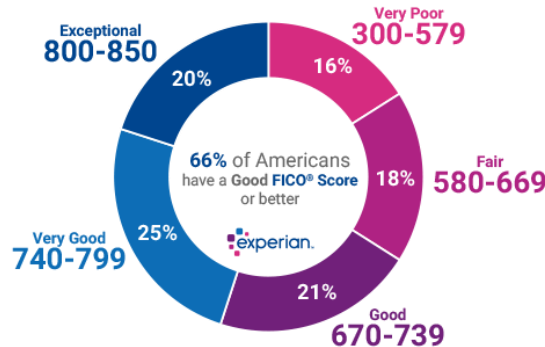
Translating a Regression Model

Assessment of probability of liquidity and full credit repayment

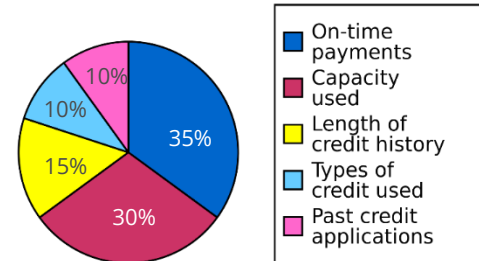
- Multiple input factors like payment history or zip code
- Output: A score for how likely the clients will repay their debt
- Examples: FICO Score (USA), Schufa (Germany)

FICO Score

- Introduced in 1989 by Fair, Isaac, and Company
- Compiles a score between 300 and 850 for creditworthiness
- Influences are payment history, debt burden, credit length, types of credit, and searches for credit



CREDIT SCORE FACTORS



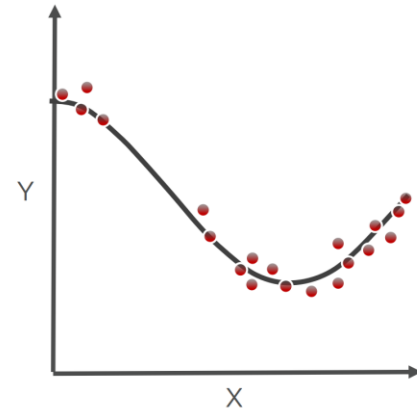
Regression Problem

Given

- A d -dimensional space D with attributes a_i , ($i = 1, \dots, d$)
- A set $V = \{v | v \in \mathbb{R}\}$ of non-discrete values
- A set $O \subseteq D$ of n observations $o = (o_1, \dots, o_n)$ with known values

Goal

- Mapping all observations $D \setminus O$ whose value is unknown
- Better understand the data



Regression Example

Training set

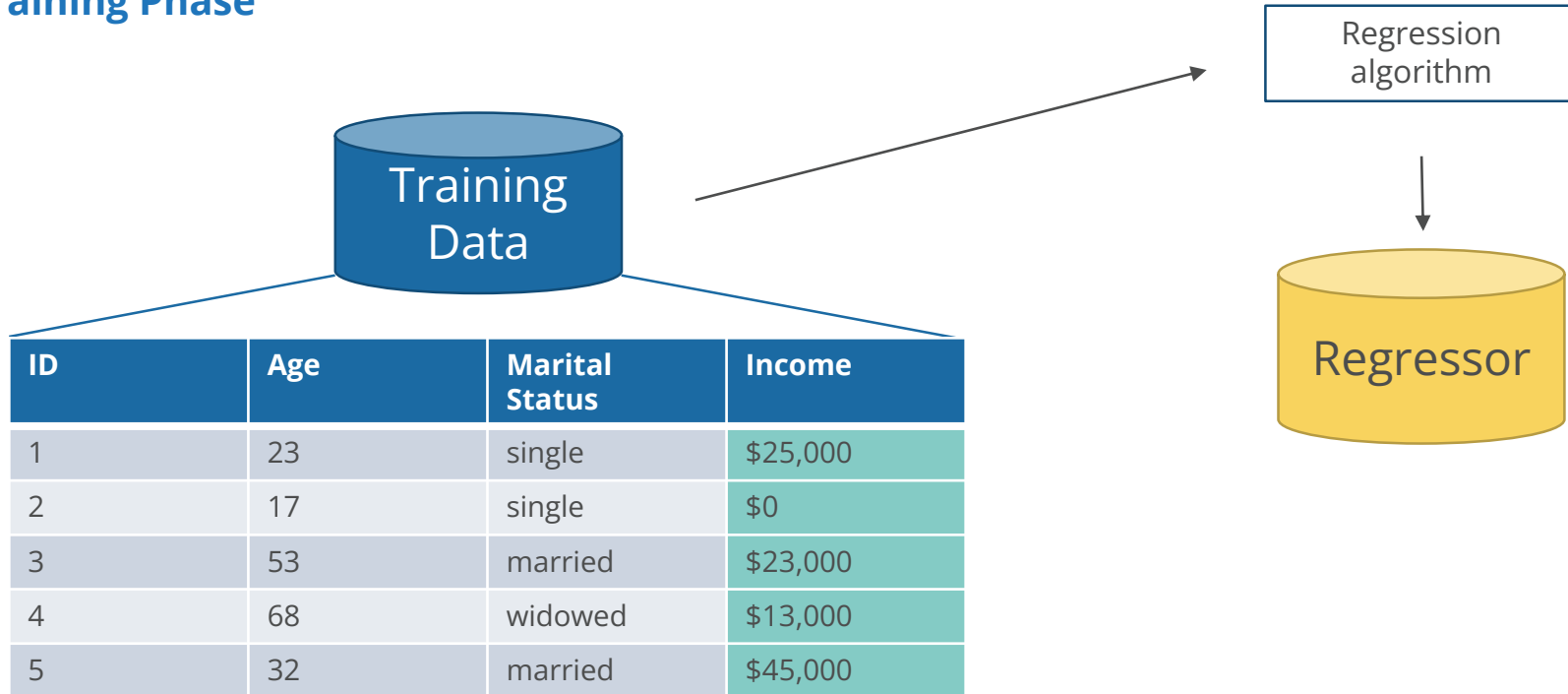
ID	Age	Marital Status	Income
1	23	single	\$25,000
2	17	single	\$0
3	53	married	\$23,000
4	68	widowed	\$13,000
5	32	married	\$45,000

Simple regressor

- Base **income** = \$25,000
- If **age** > 50 then **income** = **income** - \$12,000
- If **age** ≤ 20 then **income** = \$0
- If **marital status** = **married** then **income** = **income** + \$20,000

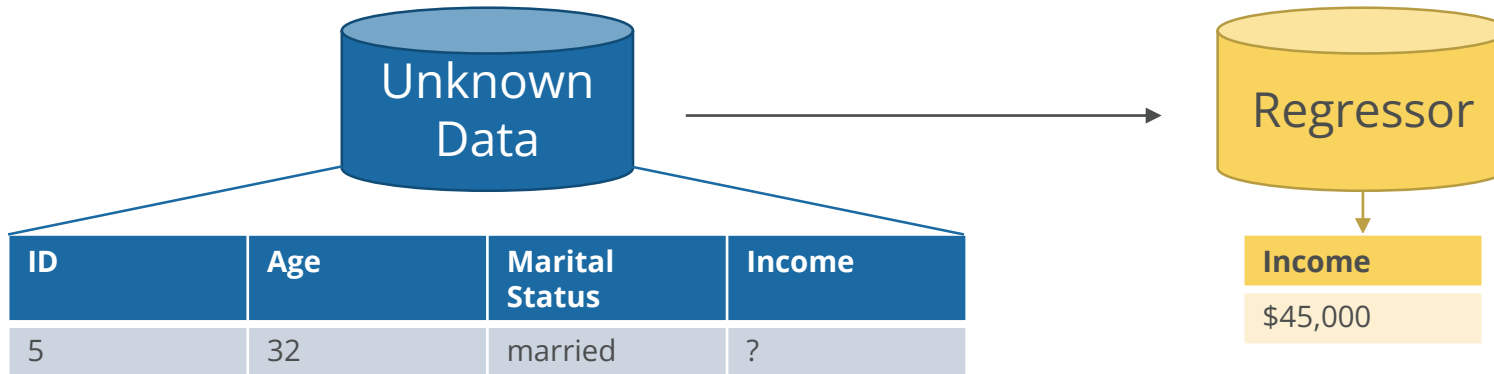
Procedure: Training

Training Phase



Procedure: Test/Forward Pass

Test phase with a single forward pass



Mean Squared Error

- Predict value of each observation $o \in O$
- Get true value for $o \in O$

$$\text{MSE} = \frac{1}{|O|} \sum_{o \in O} (\text{true value for } o - \text{predicted value for } o)^2$$

Other measurements

- Mean Absolute Error (MAE)
- [Symmetric] Mean Absolute Percentage Error ([S]MAPE)
- Root-Mean-Square Error (RMSE)

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^n |o_t - \hat{o}_t|$$

$$\text{SMAPE} = \frac{2}{n} \sum_{t=1}^n \frac{|o_t - \hat{o}_t|}{|o_t| + |\hat{o}_t|}$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^n (o_t - \hat{o}_t)^2}$$

Problem: Overfitting

- Predictor is optimized on training set (fine granularity of rules)
- Bad results on whole dataset/unknown observations
 - Quality of training set (noise, missing value, wrong values)
 - Different statistic properties of training and test set

Problem: Underfitting

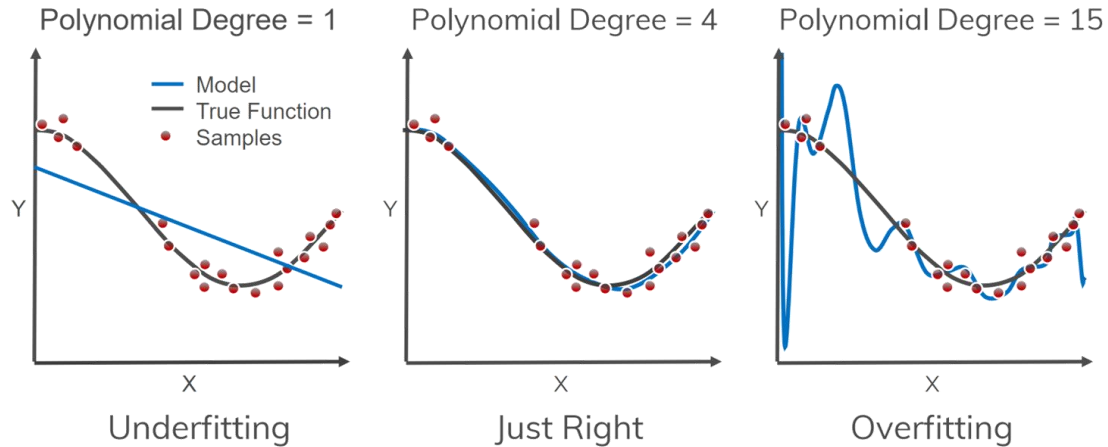
- Predictor is not optimized on training set (coarse granularity of rules)
- Bad results on whole dataset/unknown observations
 - Quality of predictor (dimensionality, parametrization)

Solution: Train-and-Test

- Split data set O into two partitions
 - Training set: Classifier learns with these observations
 - Test set: Evaluation: Prediction of class labels and comparison with existing class labels
- Problem: Not applicable if training set is too small

Evaluation of Predictors (2)

Overfitting vs Underfitting



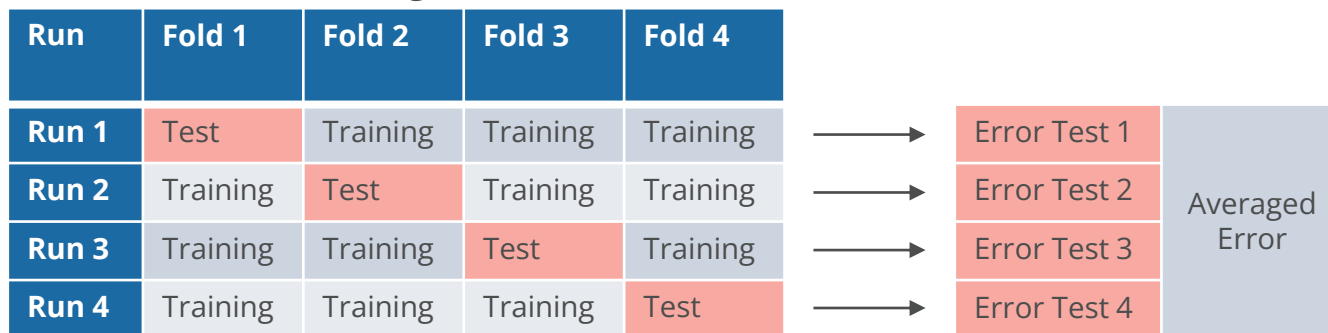
Occam's razor

- "Plurality must never be posited without necessity"
- A more general solution is always a better solution than a highly adapted solution

Evaluation of Predictors (3)

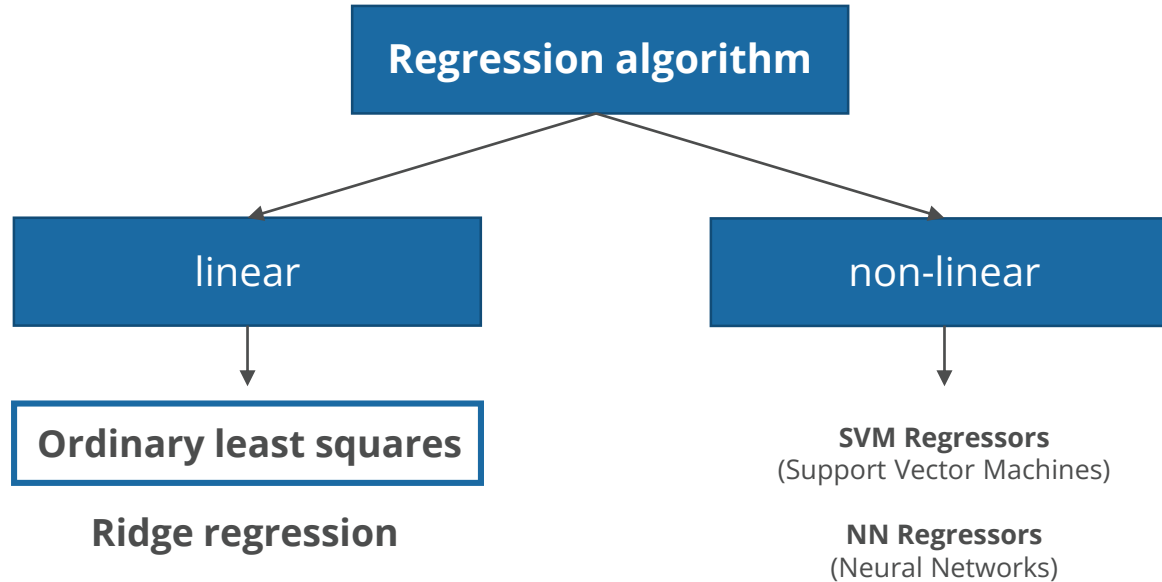
Extension: k-fold Cross Validation

- Partition dataset O into k subsets
- Training on $(k - 1)$ folds of the dataset, testing on the remaining subset
- This procedure is repeated for all k partitions, resulting in k predictors and k errors
- The k prediction errors are averaged



- Repeat with n different partitions: *n-times k-fold cross validation*
 - Compensation of bad partitions
- Variation: *Leave-one-out*: one observation for evaluation, remaining observations for training

Some Regressors

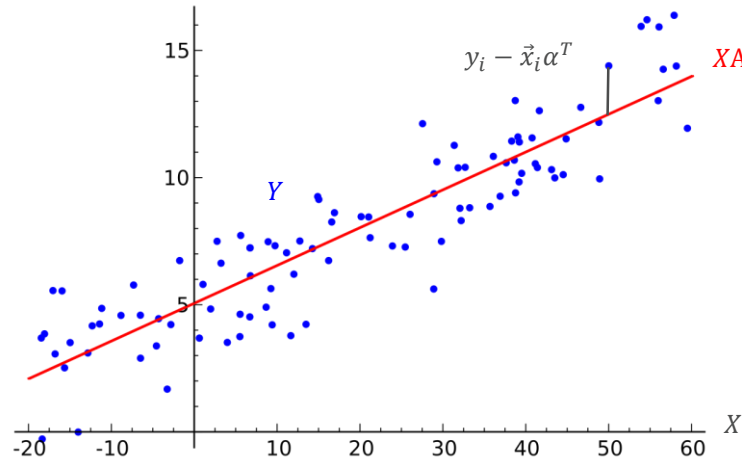


Ordinary Least Squares (OLS)

Estimator for linear regression based on minimizing the sum of squared errors (SSE) for a set of observations $O = X = \{\vec{x}_1, \dots, \vec{x}_n\}$ with $\vec{x}_i = (x_{i1}, \dots, x_{id})$ and values $V = Y = \{y_1, \dots, y_n\}$

- X : independent variable, Y : dependent variable, α : regression coefficients
- Linear model: $y_i = \alpha_1 x_{i1} + \alpha_2 x_{i2} + \dots + \alpha_d x_{id}$ or $y_i = \vec{x}_i \alpha^T$ with $\alpha = (\alpha_1, \dots, \alpha_d)$

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \vec{x}_i \alpha^T)^2$$



Ordinary Least Squares (OLS)

A simple example: real estate prices

- X : size, Y : price, α : regression coefficients
- SSE: Sum of squared errors

$$SSE = \sum_{i=1}^n ((\alpha_0 + \alpha_1 \cdot \text{size}_i) - \text{price}_i)^2$$

size	price
1000	275000
2500	370000
800	175000
1900	225000
3000	500000

$$\text{slope} = \alpha_1 = \frac{\sum_i (\text{size}_i - \overline{\text{size}})(\text{price}_i - \overline{\text{price}})}{\sum_i (\text{size}_i - \overline{\text{size}})^2} = \frac{\text{Cov}(\text{size}, \text{price})}{\text{Var}(\text{size})}$$

$$\text{intercept} = \alpha_0 = \overline{\text{price}} - \alpha_1 \overline{\text{size}}$$

Ordinary Least Squares (OLS)

Example usage of the data base

- Use of statistical aggregate functions

$$\text{slope} = \frac{\text{Cov}(\text{size}, \text{price})}{\text{Var}(\text{size})}$$

$$\text{intercept} = \overline{\text{price}} - \text{slope} \cdot \overline{\text{size}}$$

```
SELECT
    covar_samp(size, price)/var_samp(size) AS slope,
    avg(price) - (covar_samp(size, price)/var_samp(size))*avg(size) AS intercept
FROM reg;
```

size	price
1000	275000
2500	370000
800	175000
1900	225000
3000	500000



slope	intercept
118.89	90229.56

Ordinary Least Squares (OLS)

Example usage of the data base

- SQL without statistical aggregate functions

```
SELECT
  ((sumPrice * sumSizeSq) - (sumSize * sumSizePrice)) /
  ((N * (sumSizeSq)) - (sumSize * sumSize)) AS intercept,
  ((N * sumSizePrice) - (sumSize * sumPrice)) /
  ((N * sumSizeSq) - (sumSize * sumSize)) AS slope
FROM (SELECT sum(size) AS sumSize,
  sum(price) AS sumPrice,
  sum(size * size) AS sumSizeSq,
  sum(size * price) AS sumSizePrice,
  sum(price * price) AS sumPriceSq,
  count(*) AS N
FROM reg) AS a;
```

size	price
1000	275000
2500	370000
800	175000
1900	225000
3000	500000



slope	intercept
118.89	90229.56

Outline

User Defined Structures

- Data Types
- Functions (UDF)
- Stored Procedures
- Aggregate Functions

Translating a Regression Model

- Regression
- Translating OLS

User defined functions in python and R